PANTHER: Pluginizable Testing Environment for Network Protocols

Christophe Crochet [a], John Aoga [a], Axel Legay [b]

[a] *UCLouvain, {christophe.crochet, john.aoga}@uclouvain.be Belgium*
[b] *Legay Consulting, axellegay@gmail.com Belgium*

Abstract

In this paper, we introduce `PANTHER`, a modular framework for testing network protocols and formally verifying their specification. The framework incorporates a plugin architecture to enhance flexibility and extensibility for diverse testing scenarios, facilitate reproducible and scalable experiments leveraging Ivy and Shadow, and improve testing efficiency by enabling automated workflows through YAML-based configuration management. Its modular design validates complex protocol properties, adapts to dynamic behaviors, and facilitates seamless plugin integration for scalability. Moreover, the framework enables a stateful fuzzer plugin to enhance implementation robustness checks.

Keywords:
Plugins architecture, Formal Verification, Network Protocols, Network Simulation, Reproducibility, QUIC, Ivy, Black Box testing

1. Motivation and significance

Modern network protocols are vital for the reliable operation of distributed systems. However, the increasing complexity and heterogeneity of network protocols present significant challenges for testing and verification. Dynamic behaviors, time-varying properties, and the unpredictability of real-world conditions require robust methodologies that extend beyond traditional approaches like Model-Based Testing (MBT) or interoperability testing. These limitations are particularly evident in evaluating protocols' timing-sensitive features, such as congestion control and retransmissions, which demand both precision and reproducibility.

Nr.	Code metadata descryption	Please fill in this column
C1	Current code version	v1.0.3
C2	Permanent link to code/repository used for this code version	`https://github.com/ElNiak/PANTHER`
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python 3.10, docker, Ivy, Shadow
C7	Compilation requirements, operating environments and dependencies	Linux OS (Tested on Ubuntu, Debian); Docker version 27.2.1, build 9e34c9b
C8	If available, link to developer documentation/manual	`https://elniak.github.io/PANTHER`
C9	Support email for questions	christophe.crochet@uclouvain.be

Table 1: Table 1: Code metadata (mandatory)

Tools like NS2 and NS3 [9] offer robust simulation environments but lack support for executing real protocol implementations. This limits their applicability in evaluating real-world scenarios and dynamic behaviors. Frameworks like TorXakis [11] integrate formal methods for network protocol validation. However, these approaches often fail to accommodate extensibility and reproducibility, critical for testing modern protocols with diverse requirements. While tools like those used for *QUIC* testing focus on specific protocols, they lack a generalized architecture for broader applicability.

Network Simulator-centric Compositional Testing (NSCT) [10] methodology is introduced as a significant advance, integrating formal tools like `Ivy` [8, 7] with deterministic network simulators such as `Shadow` [5, 6]. NSCT showcased the effectiveness of the methodology test the real word QUIC protocol. However, NSCT's scope was limited, lacking the necessary extensibility for broader adoption and experimentation with diverse protocols. `PANTHER` [2] is a tool implementing of NSCT that overcomes these limitations.

`PANTHER` is a modular framework that combines formal verification and dynamic simulations to validate network protocol correctness. It incorporates a plugin architecture to enhance flexibility and extensibility for diverse testing scenarios, facilitates reproducible and scalable experiments leveraging Ivy and Shadow, and improves testing efficiency by enabling automated work-

flows through YAML-based configuration management. By enabling users to configure experiments according to specific needs, PANTHER allows for precise validation of both functional and non-functional properties of protocols.

The remainder of this paper is structured as follows: Section 2 provides a detailed overview of PANTHER's architecture and methodology. Section 3 presents an illustrative example, and Section 4 the impact of the tool.

2. Software description

2.1. Overview of PANTHER

PANTHER is constructed with a modular framework aimed at supporting both extensibility and reproducibility. Its main elements consist of testers, such as the adversarial testing modules generated by Ivy for formal verification; implementation components under study (IUTs) like QUIC and MiniP; execution environments that use runtime analysis tools; and network setups employing *Shadow* for deterministic simulations with precise network parameters. PANTHER incorporates Ivy, a formal verification utility that uses a domain-specific language (.ivy) to articulate protocol specifications for formal requirements. Ivy processes these formal models to produce executable C++ testers, designed as adversarial test modules that methodically explore the protocol's state space, verifying adherence to the established specifications.

The framework emphasizes reproducibility, using the Shadow network simulator to provide deterministic simulations of real-world conditions. Shadow allows precise control over parameters like latency, jitter, and bandwidth, enabling consistent experiment replication, especially for time-sensitive protocols like retransmissions and congestion control, everything encapsulated and orchestrated in a Docker container build automatically.

Additionally, PANTHER includes a self-contained Python package along with clear documentation, making it accessible and ready for use by the broader community.

2.2. Software architecture

PANTHER's plugin-based architecture enhances modularity and extensibility, supporting plugins for testing modules, Implementation Under Test (IUTs), and environments. This architecture facilitates the seamless incorporation of novel elements, such as protocol-specific IUTs or bespoke settings,

permitting PANTHER to adjust to changing testing requirements without significant re-engineering. At the core of the framework there are two main components: plugins and a configuration management system.

Plugins. PANTHER comprises three categories of plugins: services, protocols and environments. These plugins enable users to specify schemas, testing requirements, and command templates. They are dynamically incorporated via a PluginManager. Table 2 presents these plugins.

Category	**Type**	**Description**
Services	Testers	Testers generate, execute, and analyze tests for both formal verification and adversarial scenario.
	IUTs	IUT plugins define the implementation of protocols (e.g.,picoquic), its validation logic. A Jinja template is required to automate the generation of commands.
Environment	Execution	Execution environments manage runtime contexts for experiments, integrating tools like strace (tracing) and gperf (profiling).
	Network	Network environments simulate conditions using tools such as Shadow for deterministic simulations or Docker Compose for multi-container setups.
Protocol	Communication Models	Protocol plugins define communication models (e.g., client-server) and protocol-specific features (e.g., CIDs for QUIC).

Table 2: Overview of Plugins supported by *PANTHER*.

Configuration Management and Experiment Setup. The configuration management system enables users to define experiments in YAML files. Configurations specify the protocol's IUT, the tester (e.g., Ivy), the network environment (e.g., Shadow with latency and bandwidth), and execution environment settings (e.g., gperf). A ConfigLoader processes these configurations, validates them against schemas provided by plugins. The validated configurations are then dynamically processed using Jinja2 templates to generate experiment setups, such as Docker Compose files, Shadow configurations, and service launch commands, ensuring flexibility and reproducibility.

2.3. Software workflow

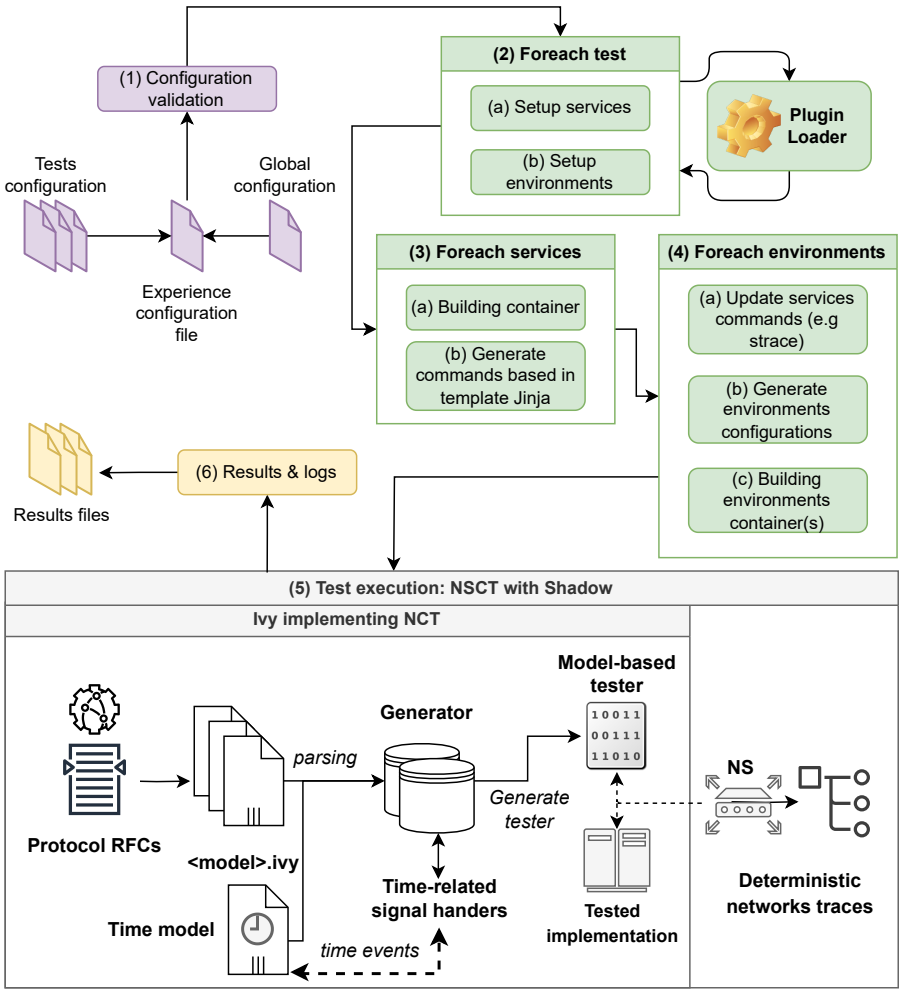

Figure 1: PANTHER workflow

The PANTHER Workflow diagram (Figure 1) outlines the process for testing network protocols. Initially, it involves validating configurations, then deploying services and environments via plugins. To maintain uniform execution, services are containerized, and dynamic service commands are crafted

using `Jinja` templates. Subsequently, environments are set up, and containers built to ensure repeatability. Protocol specifications enable Ivy to generate model-based testers, concentrating on timing aspects like retransmissions and congestion control. Using `NSCT` with the Shadow simulator guarantees deterministic network simulations and captures critical timing events during testing. Post-testing, deterministic network traces and results are provided, yielding comprehensive logs for evaluation.

3. Illustrative example

Our artifact contains many experiment configurations files demonstrating how to launch experiments with a detailed documentations. Additionally, We included many tutorials on how to add new plugins for each categories.

4. Impact

`PANTHER` uniquely combines formal verification with realistic, reproducible simulations, allowing testing of actual protocol implementations. In contrast to NS2 and NS3 [9] that lack native execution of real code, `PANTHER` uses *Shadow* to offer deterministic simulations, precisely managing network parameters like *latency* and *jitter*. Shadow accelerates formal verification and ensures time-dependent safety properties [10]. Furthermore, `PANTHER` utilizes `Ivy`'s black-box approach to validate protocols against formal specifications [4]. Its plugin architecture offers extensibility for new protocols, environments, and modules, supporting multiple protocols in a single experiment.

5. Future work

Planned upgrades for `PANTHER` involve a graphical user interface (GUI) for result visualization and experiment design, allowing intuitive visual network scenario configurations to facilitate setup and analysis. A stateful fuzzer plugin is also under development to augment `Ivy`'s formal verification, aimed at probing protocol state transitions to identify vulnerabilities and enhance implementation robustness checks. Moreover, our formal attack framework will be integrated to harmonize specifications, testing modules, and network environments, thereby improving `PANTHER`'s proficiency in validating implementations against specifications [3].

Acknoledgements. We would like to thank the belgium's "*Defence-related Research Action*" (DEFRA) and the "*Automated Methodology for Common Criteria Certification*" project (AMC3) [1].

References

[1] AMC3: What is AMC3 ? (08 2024), `https://www.amc3.be/`

[2] Crochet, C.: PANTHER: Protocol formal analysis and formal network threat evaluation resources (08 2024), `https://github.com/ElNiak/PANTHER/`

[3] Crochet, C., Aoga, J., Legay, A.: (accepted) Formally discovering and reproducing network protocols vulnerabilities. In: Nordic Conference on Secure IT Systems. Springer (2024)

[4] Crochet, C., Rousseaux, T., Piraux, M., Sambon, J.F., Legay, A.: Verifying quic implementations using ivy. Proceedings of the 2021 Workshop on Evolution, Performance and Interoperability of QUIC (2021). https://doi.org/10.1145/3488660.3493803

[5] Jansen, R., Hopper, N.J.: Shadow: Running tor in a box for accurate and efficient experimentation (2011)

[6] Jansen, R., Newsome, J., Wails, R.: Co-opting linux processes for High-Performance network simulation. In: 2022 USENIX Annual Technical Conference (USENIX ATC 22). pp. 327–350. USENIX Association, Carlsbad, CA (Jul 2022), `https://www.usenix.org/conference/atc22/presentation/jansen`

[7] McMillan, K.L., Padon, O.: Ivy: A multi-modal verification tool for distributed algorithms. Computer Aided Verification p. 190–202 (2020). https://doi.org/10.1007/978-3-030-53291-8_12

[8] Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: Safety verification by interactive generalization. ACM SIGPLAN Notices **51**(6), 614–630 (2016). https://doi.org/10.1145/2980983.2908118

[9] Riley, G.F., Henderson, T.R.: The ns-3 network simulator. In: Modeling and tools for network simulation, pp. 15–34. Springer (2010)

[10] Rousseaux, T., Crochet, C., Aoga, J., Legay, A.: Network simulator-centric compositional testing. In: Castiglioni, V., Francalanza, A. (eds.) Formal Techniques for Distributed Objects, Components, and Systems. pp. 177–196. Springer Nature Switzerland, Cham (2024)

[11] Tretmans, G., van de Laar, P.: Model-based testing with torxakis: the mysteries of dropbox revisited (2019)